Differential Evolution with Reversible Linear Transformations

Tomczak, J.M.; Węglarz-Tomczak, E.; Eiben, A.E.

Link to publication

Citation for published version (APA):
Tomczak, J. M., Węglarz-Tomczak, E., & Eiben, A. E. (2020). Differential Evolution with Reversible Linear Transformations. In *GECCO'20 Companion: proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion : July 8-12, 2020, Cancún, Mexico* (pp. 205-206). Association for Computing Machinery. https://doi.org/10.1145/3377929.3389972

Differential Evolution with Reversible Linear Transformations

Jakub M. Tomczak
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
j.m.tomczak@vu.nl

Ewelina Węglarz-Tomczak
University of Amsterdam
Amsterdam, the Netherlands
e.m.weglarz-tomczak@uva.nl

Agoston E. Eiben
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
a.e.eiben@vu.nl

ABSTRACT

Differential evolution (DE) is a well-known type of evolutionary algorithms (EA). Similarly to other EA variants it can suffer from small populations and loose diversity too quickly. This paper presents a new approach to mitigate this issue: We propose to generate new candidate solutions by utilizing reversible linear transformation applied to a triplet of solutions from the population. In other words, the population is enlarged by using newly generated individuals without evaluating their fitness. We assess our methods on three problems: (i) benchmark function optimization, (ii) discovering parameter values of the gene repressilator system, (iii) learning neural networks. The empirical results indicate that the proposed approach outperforms vanilla DE and a version of DE with applying differential mutation three times on all testbeds.

CCS CONCEPTS

• **Theory of computation → Bio-inspired optimization**;

KEYWORDS

Black-box optimization, reversible computation, population-based algorithms

1 INTRODUCTION

Optimization is about finding a solution that minimizes (or maximizes) an objective function for given constraints, *i.e.*, possible values that solutions can take. A subset of optimization problems with so called *black-box* objective functions constitute *black-box* optimization. In general, a black-box is any process that when provided an input, returns an output, but its analytical description is unavailable or it is non-differentiable [2]. Examples of black-box functions (and/or constraints) are computer programs [6], physical and biochemical processes [32, 33], or evolutionary robotics [7].

There exists a vast of derivative-free optimization (DFO) methods, ranging from classical algorithms like iterative local search or direct search [2] to modern approaches like Bayesian optimization [26] and evolutionary algorithms (EA) [3, 9]. Differential evolution

(DE) [22, 28] is one of the most successful and popular population-based DFO algorithms that utilizes evolutionary operators (mutation and crossover) and a selection mechanism to generate a new set of candidate solutions. DE is a metaheuristic with no convergence guarantees, however, it possesses multiple interesting theoretical properties [20]. Since the original publication of DE [28], the method was extended in many ways by, *e.g.*, using adaptive local search [19], modifying the differential mutation operator [35] or optimizing parameters of DE [27]. DE has been also applied to many real-life problem, such as, digital filter design [14], parameter estimation in ODEs [34], discovering predictive genes in microarray data [30], or robot navigation [18].

In this paper, we follow this line of research and present an extension of DE. One of potential issues with DE is, similarly to other EA variants, that it can suffer from small populations and loose diversity too quickly. Therefore, a potential solution is adjusting or modifying the population size [8]. Here, we propose to enlarge the population *on-the-fly* by generating new candidate solutions using reversible linear transformation applied to a triplet of solutions from the population. As a result, we take a population of N individuals and generate $3N$ new candidates assuming that we can afford running extra evaluations. This procedure allows to enhance DE and explore/exploit the search space better. We evaluate our approach on three testbeds. First, we present results on benchmark function optimization (Griewank, Rastrigin, Schwefel and Salomon functions). Second, we apply the proposed methods to discovering parameter values of the gene repressilator system. Lastly, we utilize the new DE schema in learning neural networks on image data. In all experiments, we show that enlarging the population size indeed allow to faster convergence (in terms of the number of fitness evaluations) and the reversible linear transformations provide efficient and effective alternative to the vanilla differential mutation.

The contribution of the paper is threefold. First, we propose to enhance DE by applying reversible linear transformations with two different linear operators. Second, we analyze the operators by inspecting their eigenvalues. Third, we show empirically on problems with the number of variables ranging from 4 to 4120 that the proposed DE with reversible linear transformations significantly outperforms the DE and its extension with three perturbations.

2 BACKGROUND

2.1 Black-box optimization

We consider an optimization problem of a function $f : \mathbb{X} \rightarrow \mathbb{R}$, where $\mathbb{X} \subseteq \mathbb{R}^D$ is the search space. In this paper we focus on the minimization problem, namely:

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathbb{X}} f(\mathbf{x}). \tag{1}$$

Further, we assume that the analytical form of the function f is unknown or cannot be used to calculate derivatives, however, we

can query it through a simulation or experimental measurements. Problems of this sort are known as *black-box optimization problems* [2, 13]. Additionally, we consider a bounded search space, *i.e.*, we include inequality constraints for all dimensions in the following form: $l_d \leq x_d \leq u_d$, where $l_d, u_d \in \mathbb{R}$ and $l_d < u_d$, for $d = 1, 2, \ldots, D$.

2.2 Differential Evolution

One of the most widely-used methods for black-box optimization problems is *differential evolution* (DE) [28] that requires a population of candidate solutions, $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, to iteratively generate new query points. A new candidate is generated by randomly picking a triple from the population, $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{X}$, and then $\mathbf{x}_i$ is perturbed by adding a scaled difference between $\mathbf{x}_j$ and $\mathbf{x}_k$, that is:

$$\mathbf{y} = \mathbf{x}_i + F(\mathbf{x}_j - \mathbf{x}_k), \tag{2}$$

where $F \in \mathbb{R}_+$ is the scaling factor. This operation could be seen as an adaptive *mutation operator* that is widely known as *differential mutation* [22].

Further, the authors of [28] proposed to sample a binary mask $\mathbf{m} \in \{0, 1\}^D$ according to the Bernoulli distribution with probability $p = P(m_d = 1)$ shared across all D dimensions, and calculate the final candidate according to the following formula:

$$\mathbf{v} = \mathbf{m} \odot \mathbf{y} + (1 - \mathbf{m}) \odot \mathbf{x}_i, \tag{3}$$

where $\odot$ denotes the element-wise multiplication. In the evolutionary computation literature this operation is known as *uniform crossover operator* [3, 9]. In this paper, we fix $p = 0.9$ following general recommendations in literature [21] and use the uniform crossover in all methods.

The last component of a population-based method is a selection mechanism. There are multiple variants of selection [3, 9], however, here we use the "survival of the fittest" approach, *i.e.*, we combine the old population with the new one and select N candidates with highest fitness values (*i.e.*, the deterministic $(\mu + \lambda)$ selection).

This variant of DE is referred to as "DE/*rand*/1/*bin*", where *rand* stands for randomly selecting a base vector, *1* is for adding a single perturbation (a vector difference) and *bin* denotes the uniform crossover. Sometimes it is called *classic DE* [22].

3 OUR APPROACH

Generating new candidates in DE requires sampling a triplet of solutions and, basing on these points, one solution is perturbed using the other two solutions. This approach possesses multiple advantages, naming only a few:

(i) it is non-parametric, *i.e.*, contrary to evolutionary strategies [4], no assumption on the underlying distribution of the population is made;

(ii) it has been shown to be effective in many benchmark optimization problems and real-life applications [22].

However, the number of possible perturbations is finite and relies entirely on the population size. Therefore, a small population size could produce insufficient variability of new candidate solutions. To counteract this issue, we propose the following solutions:

(1) In order to increase variability, we can perturb candidates multiple times by running the differential mutation more than once (*e.g.*, three times).

(2) In fact, we can use the selected triple of points and use it three times to generate new points. In other words, we notice that there is no need to sample three different triplets.

(3) We propose to modify the selected triplet by using generated new solutions *on-the-fly*. This approach allows to enlarge the population size.

In the following subsections, we outline the three approaches. Further, we notice that the second and the third method could be represented as linear transformations. As such, we could analyze them algebraically.

3.1 Differential Evolution x3

In the first approach we generate a larger new population by perturbing the point $\mathbf{x}_i$ using multiple candidate solutions, namely, $\mathbf{x}_j, \mathbf{x}_k, \mathbf{x}_l, \mathbf{x}_m, \mathbf{x}_n, \mathbf{x}_q \in \mathcal{X}$. Then, we can produce $3N$ new candidate solutions instead of N as follows:

$$\mathbf{y}_1 = \mathbf{x}_i + F(\mathbf{x}_j - \mathbf{x}_k) \tag{4}$$
$$\mathbf{y}_2 = \mathbf{x}_i + F(\mathbf{x}_l - \mathbf{x}_m) \tag{5}$$
$$\mathbf{y}_3 = \mathbf{x}_i + F(\mathbf{x}_n - \mathbf{x}_q). \tag{6}$$

This approach requires sampling more pairs and evaluating more points, however, it allows to better explore the search space. We refer to this approach as *Differential Evolution* ×3, or DEx3 for short.

3.2 Antisymmetric Differential Evolution

We first notice that in the DEx3 approach we sample three pairs of points to calculate perturbations. Since we pick them at random, we propose to sample three candidates $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k \in \mathcal{X}$ and calculate perturbations by changing their positions only, that is:

$$\begin{aligned}\mathbf{y}_1 &= \mathbf{x}_i + F(\mathbf{x}_j - \mathbf{x}_k) \\ \mathbf{y}_2 &= \mathbf{x}_j + F(\mathbf{x}_k - \mathbf{x}_i) \\ \mathbf{y}_3 &= \mathbf{x}_k + F(\mathbf{x}_i - \mathbf{x}_j).\end{aligned} \tag{7}$$

In other words, we perturb each point by using the remaining two. Interestingly, we notice that Eq. 7 corresponds to applying a linear transformation to these three points. For this purpose, we rewrite (7) using matrix notation by introducing matrices $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3]^\top$ and $\mathbf{X} = [\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k]^\top$ that yields:

$$\mathbf{Y} = \mathbf{MX}, \tag{8}$$

where:

$$\mathbf{M} = \begin{bmatrix} 1 & F & -F \\ -F & 1 & F \\ F & -F & 1 \end{bmatrix}. \tag{9}$$

The matrix $\mathbf{M}$ can be further decomposed as follows:

$$\mathbf{M} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{I}} + \underbrace{\begin{bmatrix} 0 & F & -F \\ -F & 0 & F \\ F & -F & 0 \end{bmatrix}}_{\mathbf{A}}, \tag{10}$$

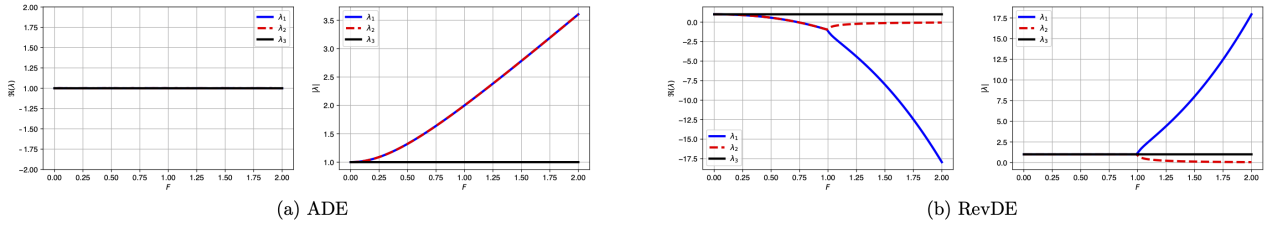where $\mathbf{I}$ denotes the identity matrix, and $\mathbf{A}$ is the antisymmetric matrix.

(a) ADE

(b) RevDE

Figure 1: Real part of eigenvalues, $\mathfrak{R}(\lambda)$, and sbsolute value of eigenvalues, $|\lambda|$, for: (a) M in ADE, and (b) R in RevDE.

Comparing Eq. 7 to DE×3 we notice that there is no need to sample additional candidates beyond one triplet. Moreover, the new mutation in (7) allows us to analyze the transformation from the algebraic perspective. Additional interesting property following from representing DE using a linear operator corresponds to parallelization of calculations and, thus, it could greatly speed up computations.

We refer to this version of DE as *Antisymmetric Differential Evolution* (ADE), because the linear transformation consists of the identity matrix and the antisymmetric matrix parameterized with the scaling factor F.

3.3 Reversible Differential Evolution

The linear transformation presented in Eq. 7 allows to utilize the triplet $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ to generate three new points, however, it could be still seen as applying DE three times, but in a specific manner (*i.e.*, by defining the linear operator $\mathbf{M}$). A natural question arises whether a different transformation could be proposed that allows *better* exploitation and/or exploration of the search space. The mutation operator in DE perturbs candidates using other individuals in the population. As a result, having too small population could limit exploration of the search space. In order to overcome this issue, we propose to modify ADE by using newly generated candidates *on-the-fly*, that is:

$$
\begin{aligned}
\mathbf{y}_1 &= x_i + F(\mathbf{x}_j - \mathbf{x}_k) \\
\mathbf{y}_2 &= x_j + F(\mathbf{x}_k - \mathbf{y}_1) \\
\mathbf{y}_3 &= x_k + F(\mathbf{y}_1 - \mathbf{y}_2).
\end{aligned}
\tag{11}
$$

Using new candidates $\mathbf{y}_1$ and $\mathbf{y}_2$ allows to calculate perturbations using points outside the population. This approach does not follow a typical construction of an EA where only evaluated candidates are mutated. Further, similarly to ADE, we can express (11) as a linear transformation $\mathbf{Y} = \mathbf{R}\mathbf{X}$ with the following linear operator:

$$
\mathbf{R} = \begin{bmatrix} 1 & F & -F \\ -F & 1 - F^2 & F + F^2 \\ F + F^2 & -F + F^2 + F^3 & 1 - 2F^2 - F^3 \end{bmatrix}.
\tag{12}
$$

In order to obtain the matrix $\mathbf{R}$, we need to plug $\mathbf{y}_1$ to the second and third equation in (11), and then $\mathbf{y}_2$ to the last equation in (11).

We refer to this version of DE as *Reversible Differential Evolution* (RevDE), because the linear transformation is reversible (see next subsection).

3.4 Algebraic properties of ADE and RevDE

3.4.1 Reversibility. An interesting property of the matrices $\mathbf{M}$ and $\mathbf{R}$ in ADE and RevDE, respectively, is that they are nonsigular matrices (see the Appendix for the proofs). Since they are non-singular, they are also invertible, and, thus, ADE and RevDE use reversible linear transformations.

The reversibility is an important property for formulating Markov Chain Monte Carlo methods (MCMC) [1]. Therefore, we could take advantage of the proposed reversible linear transformations and extend the existing work on utilizing DE for sampling methods [31]. However, this is beyond the scope of this paper and we leave investigating it in the future work.

3.4.2 Analysis of eigenvalues. We can obtain an insight into a linear operator by analysing its eigenvalues that tell us how a matrix transforms an object [29]. Therefore, they play a crucial role in analyzing properties of linear operators, *e.g.*, in control theory real parts of eigenvalues are used to determine stability of linear dynamical systems (if real part of all eigenvalues are lower than 0, then the system is stable; otherwise it is unstable [5]). Further, the absolute value of an eigenvalue λ_i determines the influence of the corresponding eigenvector [29]. If the absolute value of the eigenvector is lower than 1, then the eigenvector is a decaying mode. Similarly, if $|\lambda_i| > 1$, then the eigenvector is a dominant mode. In the case of $|\lambda_i| = 1$ we call its eigenvector a steady state.

In Figure 1 we present the absolute value and the real part of eigenvalues for $\mathbf{M}$ and $\mathbf{R}$. We notice the following facts:

- For ADE, all real parts of eigenvalues are above or equal 1, and all absolute values of eigenvalues are equal 1. As a result, the method will never lead to a decaying mode, and as such it will encourage exploration of the search space.
- For RevADE, the situation is different, namely, for $F < 0.75$ all real parts and all absolute values of eigenvalues are positive, while for $F > 0.75$ one eigenvalue has a real part equal 1 and the other two eigenvalues have real parts lower than 0. However, in all cases, all absolute values of eigenvalues are larger than 0.[1] In other words, RevDE for some values of F possesses steady states, but for $F > 1$ one eigenvalue blows up and leads to the dominant mode, while the other eigenvalue decays to 0 resulting in a decaying mode.

This analysis suggests that, in the case of ADE, taking too large F could result in generating candidate solutions that are dominated by a direction indicated by one of two eigenvectors. Consequently,

---

[1]This fact follows from the non-singularity of the matrix $\mathbf{R}$, *i.e.*, a matrix is non-singular iff all its eigenvalues are non-zero.

this could lead to "jumping" in the search space. Since ADE is closely related to DE×3, this result sheds an additional light on the behavior of DE, and seems to confirm that taking F larger than 0.5 in DE is not a reasonable decision.

In the case of RevDE it seems that taking values of F below 0.75 is preferable, because then the linear operator will not lead to dominating modes. As a result, a better exploitation/exploration of the search space could be achieved.

4 EXPERIMENTS

In order to verify our approach empirically, we compare the three proposed methods and the standard DE on three testbeds:

(1) *Benchmark functions*: selected benchmark function for optimization.
(2) *Gene Repressilator System*: discovering parameter values of a system of ordinary differential equations for given observations.
(3) *Neural Networks Learning*: learning a neural network with one hidden layer on image dataset.

In all experiments, we used the uniform crossover with $p = 0.9$ for all methods. The scale parameter F was selected from the following range of values: $\{0.125, 0.25, 0.375, 0.5, 0.6, 0.625, 0.675, 0.75\}$. The population size was set to 500 across all experiments. We pick the base vector *randomly*.

The code of the methods and all experiments is available under the following link: https://github.com/jmtomczak/reversible-de.

4.1 Benchmark Function Optimization

4.1.1 Details. We evaluate the proposed methods on the optimization task of four benchmark functions:

- Griewank function [11]:

$$f(\mathbf{x}) = 1 + \sum_{d=1}^{D} \sqrt{x_d^2/4000} - \prod_{d=1}^{D} \cos\left(\frac{x_d}{\sqrt{d}}\right) \tag{13}$$

with the box constraints: $\forall_{d \in \{1, 2, ..., D\}} \ x_d \in [-5, 5]$;

- Rastrigin function [23]:

$$f(\mathbf{x}) = 10D + \sum_{d=1}^{D} \left(x_d^2 - 10 \cos\left(2\pi x_d\right)\right) \tag{14}$$

with the box constraints: $\forall_{d \in \{1, 2, ..., D\}} \ x_d \in [-5, 5]$;

- Salomon function [24]:

$$f(\mathbf{x}) = 1 - \cos\left(2\pi \sqrt{\sum_{d=1}^{D} x_d^2}\right) + 0.1 \sqrt{\sum_{d=1}^{D} x_d^2} \tag{15}$$

with the box constraints: $\forall_{d \in \{1, 2, ..., D\}} \ x_d \in [-5, 5]$;

- Schwefel function [25]:

$$f(\mathbf{x}) = 418.9829D - \sum_{d=1}^{D} \left(x_d \sin\left(\sqrt{|x_d|}\right)\right) \tag{16}$$

with the box constraints: $\forall_{d \in \{1, 2, ..., D\}} \ x_d \in [200, 500]$.

We test the methods on these function with different dimensions, namely, $D \in \{10, 30, 100\}$. We run DE×3, ADE and RevDE for 150 generations. Since DE evaluates three times less candidate solutions, we run it for 450 generations to match the number of evaluations.

However, we want to highlight that DE is more *informed* than other methods due to the propagation of new solutions in consecutive iterations (*i.e.*, applying the selection mechanism 3 times more). All experiments are repeated 10.

4.1.2 Results & Discussion. The results of the best solution found until given evaluation for this experiment are presented in Figure 2. First, we notice that ADE and DEx3 similarly to the DE. However, in 3 out of 12 cases (*i.e.*, Griewank with $D = 100$), Rastrigin with $D = 100$) and Schwefel with $D = 100$) DE is able to converge to a better solution than ADE and DEx3. Nevertheless, the results are similar and in the next experiments we skip comparing to DE, because it is not completely fair due to the difference in the number of generations.

Interestingly, ADE performs almost identically as DEx3. This result seems to confirm that it is unnecessary to sample multiple (*i.e.*, three) triplets, and utilizing a single triplet to generate new candidates is sufficient.

In all test cases, RevDE achieved the best results in terms of both final objective value and convergence speed. This result is remarkable, because new candidate solutions are generated *on-the-fly* and are used to generate to new points. Moreover, for $D = 30$ and $D = 100$, *i.e.*, the higher-dimensional cases, RevDE outperformed other methods significantly. These results are especially promising for real-life applications like parameter values discovery of mechanistic models and computer programs [6] or learning controllers in (evolutionary) robotics [15, 16].

In the Rastrigin function with $D = 30$ and $D = 100$ there is a peculiar behavior of RevDE where around the evaluation number 80000 and 50000, respectively, there is a large improvement in terms of the objective value. We hypothesize that the optimizer "jumps out" from a local minimum due to large eigenvalue as discussed in Section 3.4.

4.2 Gene Repressilator System

4.2.1 Details. The gene repressilator system proposed in [10] is a popular model for gene regulatory systems and consists of three genes connected in a feedback loop, where each gene transcribes the repressor protein for the next gene in the loop. The model consists of six ordinary differential equations that describe dependencies among mRNA (m_1, m_2, m_3) and corresponding proteins (p_1, p_2, p_3), and four parameters $\mathbf{x} = [\alpha_0, n, \beta, \alpha]^\top$, which are as follows:

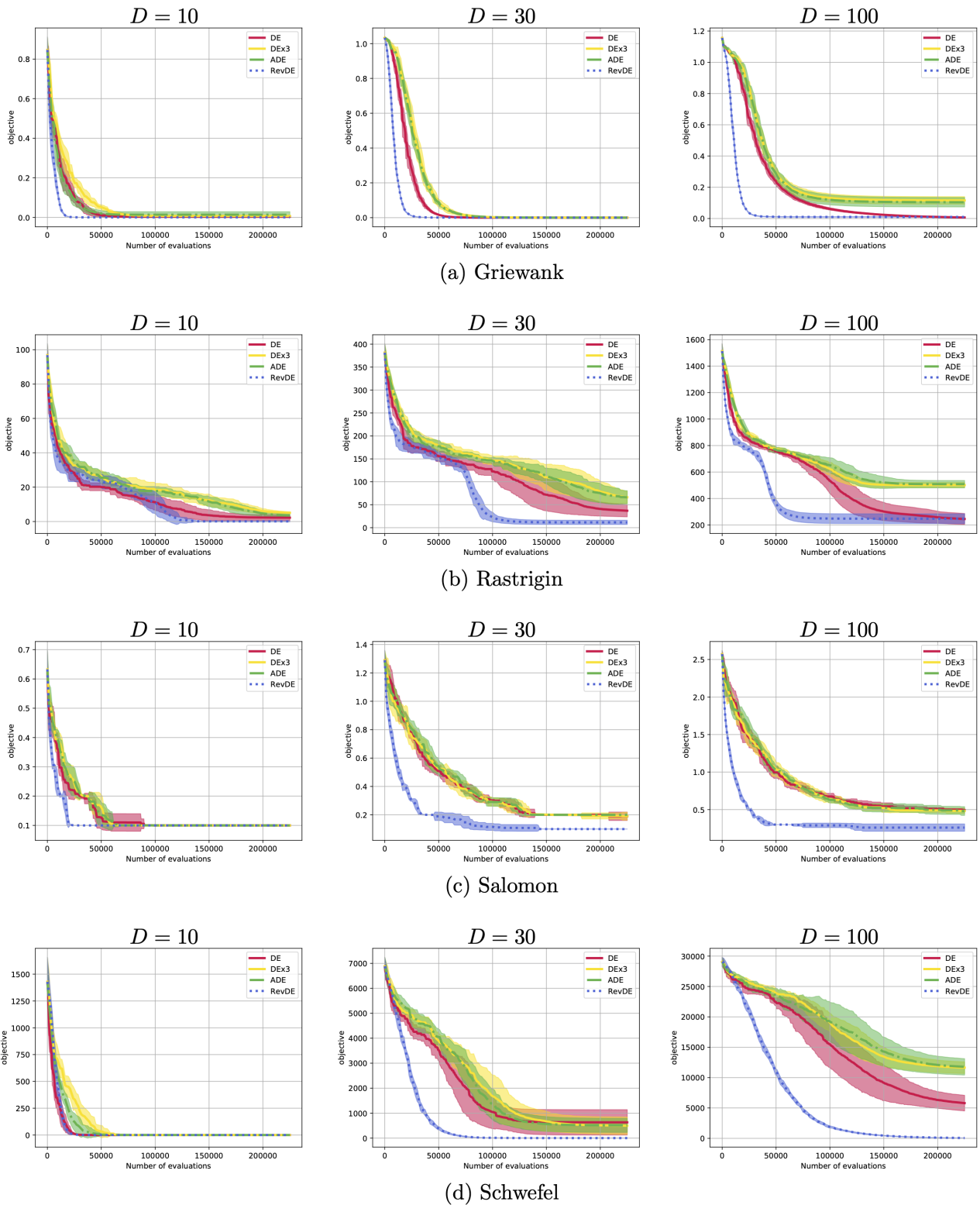$$\frac{dm_1}{dt} = -m_1 + \frac{\alpha}{1 + p_3^n} + \alpha_0 \tag{17}$$

$$\frac{dp_1}{dt} = -\beta(p_1 - m_1) \tag{18}$$

$$\frac{dm_2}{dt} = -m_2 + \frac{\alpha}{1 + p_1^n} + \alpha_0 \tag{19}$$

$$\frac{dp_2}{dt} = -\beta(p_2 - m_2) \tag{20}$$

$$\frac{dm_3}{dt} = -m_3 + \frac{\alpha}{1 + p_2^n} + \alpha_0 \tag{21}$$

$$\frac{dp_3}{dt} = -\beta(p_3 - m_3). \tag{22}$$

Figure 2: The results of the best solution found until given evaluation on four benchmark black-box optimization testbeds (a) Griewank function, (b) Rastrigin function, (c) Salomon function, (d) Schwefel function, and three cases: (left column) 10D case, (middle column) 30D case, (right column) 100D case. The solid red lines correspond to DE, the solid yellow lines are for DEx3, the dotted-dashed green lines depict ADE, and the dotted blue lines represent RevDE. In all 12 test cases an average and one standard deviations over 10 runs are presented.
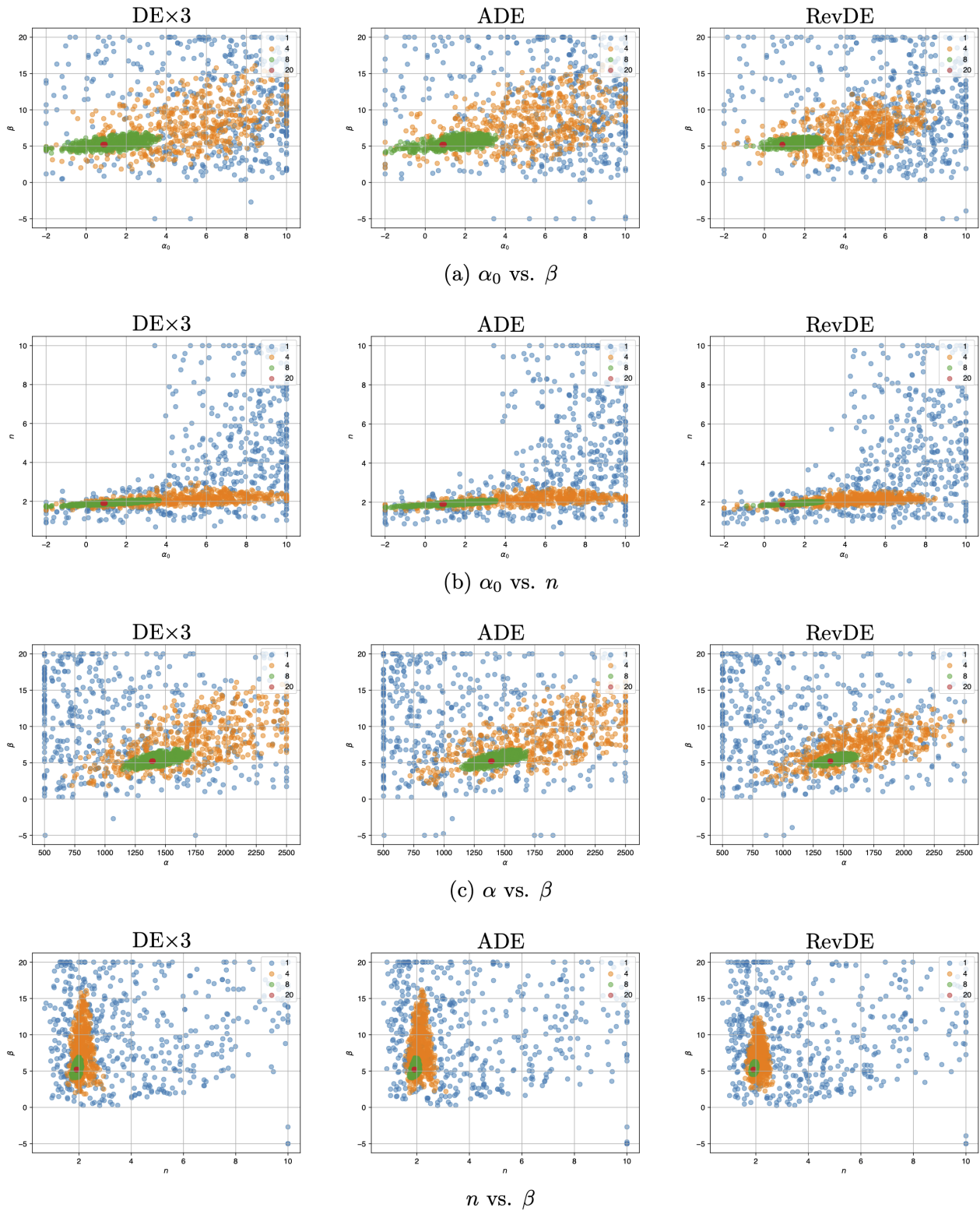
(a) α_0 vs. β

(b) α_0 vs. n

(c) α vs. β

n vs. β

Figure 3: The discovered parameter values in the repressilator model by: (*left column*) DEx3, (*middle column*) ADE, (*right column*) RevDE. Colors of dots represent the generation number: blue is the 1st generation, orange is the 4th generation, green is the 8th generation, red is the 20th generation. The real parameter values: $(\alpha_0, n, \beta, \alpha) = (1, 2, 5, 1000)$.

Further, we assume that only mRNA measurement are measured, while proteins are considered as missing data. The **goal** of this experiment is to discover the parameters' values for a given observation of mRNA. We transform this problem into the minimization of the following objective:

$$f(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \sqrt{\sum_{i=1}^{3} \left(m_{i,n} - m_{i,n}(\mathbf{x}) \right)^2}, \qquad (23)$$

where $m_{i,n}(\mathbf{x})$ is given by numerically integrating the system of differential equations in (17–22) using a solver, *e.g.*, a Runge-Kutta method[2]. Notice that the objective function is black-box due to the non-differentiable simulator.

We follow the settings outlined in [33]. The real parameters' values are assumed to be $\mathbf{x}^* = [1, 2, 5, 1000]^\top$ and we generate real values of m_i by first solving the equations (17–22) with $\mathbf{x}^*$ and given initial conditions $(m_1, p_1, m_2, p_2, m_3, p_3) = (0, 2, 0, 1, 0, 3)$, and then adding Gaussian noise with the mean equal 0 and the standard deviation equal 5.

We run all methods for 20 generations. All experiments were repeated 10 times. For analyzing final solutions, we look into the convergence of a population from a single run.
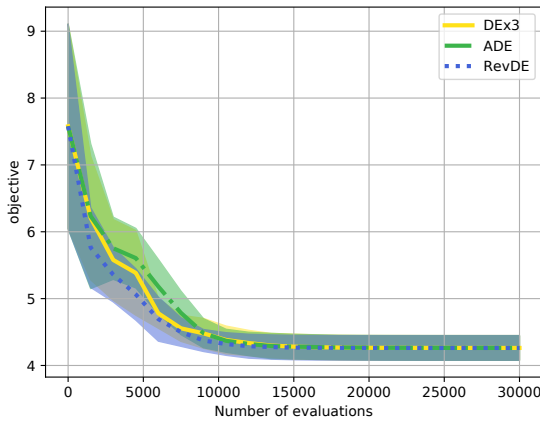
Figure 4: The results of the best solution found until given fitness evaluation on the repressilator model. The average and the standard deviation over 10 runs are reported.

4.2.2 Results & Discussion. We present results of the best solution found until given objective evaluation in Figure 4. Further, we depict converging process of the population in Figure 3. We present only a single run out of ten, however, the behavior is almost indistinguishable between runs.

All methods achieve almost identical objective values (see Figure 4). RevDE seems to converge slightly faster (on average) than other methods, but the difference is not significant.

We can obtain more insight into the performance of the methods by analysing behavior of the population over generations (see Figure 3). First we notice that all methods converge to a single point

―――――――――
[2]In this work, we used the explicit Runge-Kutta method of order 5(4) provided by SciPy: https://www.scipy.org/.

within 20 generations. Second, it seems that ADE and DEx3 behave almost indistinguishably. Comparing their scatterplots it is almost impossible to spot a difference. However, RevDE converges faster, because already in the 4th generation the solutions are less scattered than in the case of DEx3 and ADE. In other words, comparing how points are distributed in the 4th and the 8th generation, it is apparent that the variance for the populations found by RevDE is smaller than for ADE ans DEx3.

Eventually, we would like to comment on discovering the parameter values. In the case of ADE, DEx3 and RevDE all candidates converge to the same solution that are roughly around the point $\mathbf{x} = [1, 2, 5, 1380]^\top$. By comparing the real values and the discovered ones we see that the only mismatch is for α. However, the discrepancy (1000 vs. 1380) possibly follows from the fact that the observed data is noisy, because the population in the 4th epoch covered values around 1000 well. Similarly, in [33] the Approximate Bayesian Computation with the Sequential Monte Carlo method (ABC-SMC) also obtained values between around 800 and 1300 (see Figure 4(c) in [33]). We conclude that RevDE seems to be a very promising alternative to Monte Carlo techniques for finding parameters in simulator-based inference problems [6].

4.3 Neural Networks Learning

4.3.1 Details. In the last experiment we aim at evaluating our approach on a high-dimensional optimization problem. For this purpose, we train a neural network with a single fully-connected hidden layer on the image dataset of ten handwritten digits (MNIST [17]). We resize original images from 28px × 28px to 14px × 14px, and use 20 hidden units. As a result, we obtain the total number of weights equal 4120 (*i.e.*, $\mathbb{X} = \mathbb{R}^{4120}$). We use the ReLU non-linear activation function for hidden units and the softmax activation function for outputs. The objective function is the classification error:

$$f(\mathbf{x}) = 1 - \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[y_n = y_n(\mathbf{x})], \qquad (24)$$

where N denotes the number of images, $\mathbb{I}[\cdot]$ is an indicator function, y_n is the true label of the nth image, and $y_n(\mathbf{x})$ is the label for the nt image predicted by a neural network with weights $\mathbf{x}$. The prediction of the neural network is a class label with highest value given by the softmax output.

The original dataset consists of 60000 pairs of images and labels for training, and 10000 pairs of images and labels for training. In our experiments, we use only 2000 training points, but all 10000 testing points. All models are trained for 500 epochs (generations) and the experiments are repeated 3 times. For testing, we take a candidate solution from the final population with the lowest training classification error.

The objective function in Eq. 24 is non-differentiable, and, thus, could be treated as a black-box objective. However, we want to highlight that this experiment does not aim at proposing DE as an alternative training procedure to a gradient-based methods, because the log-likelihood function is a good proxy to the objective in (24). In fact, is has been shown in multiple papers that the (stochastic) gradient descent optimizer is extremely effective in learning neural networks and DE is not competitive with it at all [12]. We rather use

the neural network learning problem as an interesting showcase of a high-dimensional optimization problem.

Table 1: Test results on MNIST. The average with the standard errors over 3 runs are reported.

Method	Classification error
DE×3	20.1 ± 1.4
ADE	18.1 ± 0.2
RevDE	18.5 ± 0.8

4.3.2 Results & Discussion. In Figure 5 we present learning curves for neural networks trained with different methods. Additionally, in Table 1 we gather test classification errors.

First of all, we notice that the training is not fully converged and possibly better results could have been achieved. Nevertheless, our goal is to present performance of our methods on a high-dimensional problem rather than reaching state-of-the art scores. That being said, we first observe that the proposed extensions of DE shared similar learning curves. ADE performed the best during training, and RevDE converged to a better point than DEx3 in the very end. However, the final test performance was better for ADE and RevDE than DEx3. This result could be possibly explained by the fact that DEx3 is more stochastic than the other two methods that could be harmful in the highly dimensional problem.

A close inspection of the results in Table 1 suggests that ADE and RevDE perform on par, and they seem to be better than DEx3. This result is potentially interesting because the negative message delivered in [12] that DE is definitely worse than the gradient-based learning method is not necessarily true and more research in this direction is required. Especially in the context of adding non-differentiable components (regularizers) to the learning objective.
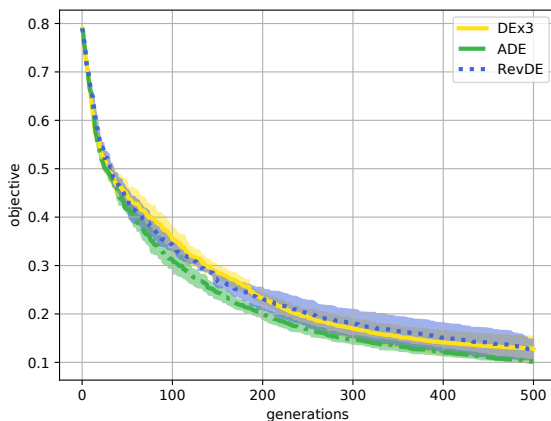
Figure 5: Training curves on MNIST. The average and the standard deviation over 3 runs are reported.

5 CONCLUSION

In this paper, we note that insufficient variability of the population could cause DE to loose diversity too quickly. In order to counteract this issue, we propose three extensions of DE: (i) DE with multiple samples of candidates for calculating perturbations, (ii) DE with the reversible linear transformation using a sum of the identity matrix and an anti-symmetric matrix, (iii) DE with the reversible linear transformation utilizing newly generated yet not evaluated candidates.

We provide a theoretical analysis of the proposed linear operators by proving their reversibility, and inspecting their eigenvalues. Further, we show empirically on three testbeds (benchmark function optimization, discovering parameter values of the gene repressilator systems, and learning neural networks) that producing new candidates *on-the-fly* allows to obtain better results in fewer number of evaluations compared to DE.

We believe that this work opens new possible research directions:

- Representing the differential mutation as a linear transformation allows to look into other forms of linear operators.
- The linear operators defined in this paper are parameterized with a single parameter. A natural extension would be considering different parameterization.
- Here, we present an analysis based on eigenvalues. However, we can consider the reversible transformation as a dynamical system (*e.g.*, an extension of the analysis outlined in [20]).
- We can take advantage of the reversibility of the proposed linear transformations. For instance, reversibility is an important property of transition operators in MCMC methods [1]. A modification of the vanilla DE for formulating a proper MCMC method was already presented in [31] and an interesting direction would be to extend this work using DE with the reversible linear transformations.

APPENDIX

Non-singularity of M and R

PROPOSITION 5.1. *The matrix* **M** *defined in Eq. 9 is non-singular.*

PROOF. Since the matrix **M** is a small 3-on-3 matrix, we can calculate its determinant analytically that gives:

$$\det(\mathbf{M}) = 1 + 3F^2. \tag{25}$$

For any value of F we have $\det(\mathbf{M}) \neq 0$, therefore, the matrix **M** is non-singular □

PROPOSITION 5.2. *The matrix* **R** *defined in Eq. 12 is non-singular.*

PROOF. The matrix **R** is a small 3-on-3 matrix, thus, we can calculate its determinant analytically, that gives:

$$\det(\mathbf{R}) = 1. \tag{26}$$

Since the determinant is always 1, then **R** is non-singular. □

ACKNOWLEDGMENTS

REFERENCES

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I. Jordan. 2003. An introduction to MCMC for machine learning. *Machine Learning* 50, 1-2 (2003), 5–43.

[2] Charles Audet and Warren Hare. 2017. *Derivative-free and Blackbox Optimization*. Springer.

[3] Thomas Bäck. 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford university press.

[4] Thomas Bäck, Christophe Foussette, and Peter Krause. 2013. *Contemporary Evolution Strategies*. Springer.

[5] Zdzisław Bubnicki. 2005. *Modern Control Theory*. Springer.

[6] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. 2019. The frontier of simulation-based inference. *arXiv preprint arXiv:1911.01429* (2019).

[7] Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E. Eiben. 2015. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI* 2 (2015), 4.

[8] Agoston E. Eiben, Elena Marchiori, and VA Valko. 2004. Evolutionary algorithms with on-the-fly population size adjustment. In *International Conference on Parallel Problem Solving from Nature*. Springer, 41–50.

[9] Agoston E. Eiben and James E. Smith. 2015. *Introduction to Evolutionary Computing*. Vol. 53. Springer.

[10] Michael B Elowitz and Stanislas Leibler. 2000. A synthetic oscillatory network of transcriptional regulators. *Nature* 403, 6767 (2000), 335–338.

[11] Andreas O. Griewank. 1981. Generalized descent for global optimization. *Journal of optimization theory and applications* 34, 1 (1981), 11–39.

[12] Jarmo Ilonen, Joni-Kristian Kamarainen, and Jouni Lampinen. 2003. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters* 17, 1 (2003), 93–105.

[13] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13, 4 (1998), 455–492.

[14] Nurhan Karaboga. 2005. Digital IIR filter design using differential evolution algorithm. *EURASIP Journal on Advances in Signal Processing* 2005, 8 (2005), 856824.

[15] Gongjin Lan, Matteo De Carlo, Fuda van Diggelen, Jakub M. Tomczak, Diederik M. Roijers, and Agoston E. Eiben. 2020. Learning Directed Locomotion in Modular Robots with Evolvable Morphologies. *arXiv preprint arXiv:2001.07804* (2020).

[16] Gongjin Lan, Milan Jelisavcic, Diederik M. Roijers, Evert Haasdijk, and Agoston E. Eiben. 2018. Directed locomotion for modular robots with evolvable morphologies. In *International Conference on Parallel Problem Solving from Nature*. Springer, 476–487.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[18] Erasmo Gabriel Martinez-Soltero and Jesus Hernandez-Barragan. 2018. Robot navigation based on differential evolution. *IFAC-PapersOnLine* 51, 13 (2018), 350–354.

[19] Nasimul Noman and Hitoshi Iba. 2008. Accelerating differential evolution using an adaptive local search. *IEEE Transactions on evolutionary Computation* 12, 1 (2008), 107–125.

[20] Karol R Opara and Jarosław Arabas. 2019. Differential Evolution: A survey of theoretical analyses. *Swarm and evolutionary computation* 44 (2019), 546–558.

[21] Magnus Erik Hvass Pedersen. 2010. *Good parameters for differential evolution*. Technical Report HL1002. Hvass Laboratories.

[22] Kenneth Price, Rainer M Storn, and Jouni A. Lampinen. 2006. *Differential Evolution: A Practical Approach to Global Optimization*. Springer Science & Business Media.

[23] Leonard Andreevič Rastrigin. 1974. Systems of extremal control. *Nauka* (1974).

[24] Ralf Salomon. 1996. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems* 39, 3 (1996), 263–278.

[25] Hans-Paul Schwefel. 1981. *Numerical optimization of computer models*. John Wiley & Sons, Inc.

[26] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.

[27] Mudita Sharma, Alexandros Komninos, Manuel López-Ibáñez, and Dimitar Kazakov. 2019. Deep reinforcement learning based parameter control in differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 709–717.

[28] Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359.

[29] Gilbert Strang. 2016. *Introduction to Linear Algebra*. Wellesley-Cambridge Press Wellesley, MA.

[30] Dimitris K. Tasoulis, Vassilis P. Plagianakos, and Michael N. Vrahatis. 2006. Differential evolution algorithms for finding predictive gene subsets in microarray data. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 484–491.

[31] Cajo J.F. Ter Braak. 2006. A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces. *Statistics and Computing* 16, 3 (2006), 239–249.

[32] Jakub M. Tomczak and Ewelina Węglarz-Tomczak. 2019. Estimating kinetic constants in the Michaelis–Menten model from one enzymatic assay using Approximate Bayesian Computation. *FEBS letters* 593, 19 (2019), 2742–2750.

[33] Tina Toni, David Welch, Natalja Strelkowa, Andreas Ipsen, and Michael P.H. Stumpf. 2009. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface* 6, 31 (2009), 187–202.

[34] Feng-Sheng Wang, Tzu-Liang Su, and Horng-Jhy Jang. 2001. Hybrid Differential Evolution for Problems of Kinetic Parameter Estimation and Dynamic Optimization of an Ethanol Fermentation Process. *Industrial & Engineering Chemistry Research* 40, 13 (2001), 2876–2885.

[35] Jingqiao Zhang and Arthur C. Sanderson. 2009. JADE: Adaptive Differential Evolution with Optional External Archive. *IEEE Transactions on evolutionary computation* 13, 5 (2009), 945–958.